Runtime Systems for Exascale Programming Languages

Bradford L. Chamberlain, Cray Inc. (bradc@cray.com)

Larry Kaplan, Cray Inc. (lkaplan@cray.com)

This paper proposes research in a unified runtime layer that would support next-generation languages, like Chapel, for exascale systems. Whether or not Chapel becomes a preferred language for exascale computing, we believe that many of its characteristics—a partitioned global namespace, dynamic task parallelism, explicit features for controlling and reasoning about locality, high-level data parallelism with user-level control over layouts, distributions, and parallelization strategies—will be important for productive exascale computing and likely to be considered for inclusion in future exascale programming models, whatever they end up being.

Description of Research Direction

Parallel languages are typically implemented on top of runtime libraries that provide the compiler-generated code with access to system-level capabilities. For example, Chapel's runtime libraries provide support for tasking (which implements all parallelism in the language), synchronization (to coordinate between tasks), communication (to support distributed memory execution), and memory management, as well as other miscellaneous features required by the language. These capabilities are typically implemented in terms of an abstract interface that supports the ability to choose from a number of underlying implementation technologies. For example, today a Chapel user can choose between Pthreads- or Qthreads-based tasking and synchronization; GASNet- or MPI-based communication; and memory management provided by the C runtime or TCMalloc.

A key observation is that most languages that support flexible runtimes, like Chapel, couple multiple disparate technologies together to provide the full suite of runtime capabilities. As an example, GASNet and Qthreads are implemented completely independently of one another and do not share any semantic information or state when used together to implement a Chapel program. As a result, opportunities for sharing information between the tasking and communication sub-interfaces for the purposes of analysis or optimization are limited. While mixing independent technologies in this manner is arguably sufficient on today's system architectures, it leaves optimization opportunities on the table and limits the degree to which the runtime layers can proactively be involved in dynamically optimizing a user's program.

With the coming of exascale-era systems, the need for more intelligent and advanced runtimes for languages grows. Due to the increased sensitivity to locality on a chip due to greater hierarchy and/or heterogeneity, the sensitivity to data movement for energy and performance reasons, and the restricted bandwidth-to-FLOP ratio that is anticipated, involving the runtime in resource management is likely to be increasingly important. Removing the divides between traditionally disparate aspects of the runtime will go a long way toward making decisions cooperatively between layers to improve optimality. Unifying the runtime in this manner can also have a positive impact on resiliency, portability, and interoperability. To that end, the research suggested by this position paper is for *integrated* language runtimes for next-generation languages and programming models.

Our vision of an idealized unified runtime would combine the single-sided communication capabilities of a SHMEM or GASNet; the active messages of a GASNet or HPX; the portability, vendor adoption, and backwards-compatibility provided by MPICH; the user-level tasking and synchronization of a Qthreads or Nanos++; the throttling/load-balancing/work-stealing capabilities of a Habanero, Scioto, or TBB; and the integrated, scalable memory management of TCMalloc or the like. Such capabilities would need to be customizable to any likely architectural candidate for exascale computing and able to be integrated into the languages and compilers targeting that hardware in order to reflect each language's abstract machine model.

Challenges Addressed

Programmability: By enabling rich language semantics like a partitioned global namespace and active messages, programmers will be able to write code for exascale machines more easily. By providing more cross-function knowledge to the runtime, more advanced runtime capabilities will be able to shift management responsibilities from the user and compiler. To the extent that the runtime is backwards-compatible with MPICH, interoperability with existing petascale applications will be enabled. To the extent that multiple languages will be able to share the same runtime, interoperability between emerging programming models will be achieved.

Scalability/Performance Portability: Making tasking and communication aware of one another should provide rich opportunities for optimization such as latency hiding and task switching upon heavyweight communication events. This is particularly important when communication becomes more than simply "off-chip vs. on-" due to multiple realms of locality within each compute node. Performance portability will benefit by making the runtime consistent across a large number of platforms, yet tuned for each one.

Energy efficiency: By integrating memory, tasking, and communication, energy-aware decisions can be made regarding data and task placement in order to minimize data movement. By integrating tasking and communication, the tasking layer can make scheduling decisions based on expected latencies for communications and use the hardware more effectively and efficiently.

Resilience: By making communications, tasking, and memory aware of one another in a single integrated runtime, the state required to recover from a failure can be better understood across disparate parts of the runtime system, permitting key state to be saved, restored, and duplicated as necessary.

Maturity/Novelty

As a community, we have seen several compelling demonstrations of disparate communication and tasking technologies working effectively in isolation, such as MPICH, GASNet, and Qthreads. Though there have been some recent moves toward more unified runtimes by DOE labs and academic groups, to date most of these have not been particularly mature or general-purpose (across languages and architectures). To that end, we would advocate continued funding for such groups or incentives for groups with mature individual technologies to work together in order to unify and extend them.

Uniqueness

Such integrated runtimes would be useful in arenas other than exascale, though unlikely to be effectively funded outside of the HPC community due to our specialized need for distributed memory computation, scalability, and more stringent communication requirements than would be expected in the cloud/datacenter computing sectors. Within HPC, an integrated runtime like this could be beneficial prior to the exascale era, but it's likely that the complexity of exascale nodes will increase the need for, and benefits from, such integration and cross-layer optimization.

Applicability

If successful, a unified runtime like this should be broadly applicable across system scales, programming models, and target architectures.

Effort

Good progress could be made with approximately 6 FTEs for several years.